Prediction of Swarm Popularity in Peer-to-Peer Networks

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Bo Wen

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Professor Haiyang Wang

July 2019

Acknowledgements

Throughout the writing of my master thesis, I have received tremendous help from the CS department. Both my supervisor and professor inside the department. I would first like to thank my supervisor, Dr. Haiyang Wang, who helps me and guide me with the formatting and mythology of the thesis. I would thank for other master students for nice suggestions and helping me to collect the data. And my family members who support me always.

Dedication

For all the work I did, I dedicate to my family, they don't really understand what I am working with. But they still supporting me for what I am doing. For the professor and classmates during the two years of a master life.

Abstract

Recent years have witnessed the great popularity of adopting BitTorrent(BT)-like peer-to-peer(P2P) designs in Internet Application. Different from the classic client-server solutions, P2P significantly improves the capacity, robustness, and the scalability of the system. However, it is known that P2P will also bring certain Quality-of-Service(QoS) issues. In particular, users' service performance and availability can hardly be guaranteed. To make the matter worse, the users cannot even predict their service performance, such as the downloading rate and completion time, before joining the swarm.

In this thesis, we aim to provide an option to help the users predict their service performance in BitTorrent-like P2P applications. Using the classic BT as a case study, we collect 1250 torrent files and carefully investigate their content, peer, and service performance details. Our measurement shows that the swarm popularity is related to the file type, file size, piece length, and its creation time. To better capture this relationship, we adopt different prediction methods to help the users better understand their possible downloading performance. Our evaluation shows that the neural-network-based approach has the best accuracy with minimal cost. In the future, we are aiming to further optimize the accuracy and implement the prototype of our approach in the existing open-source BT releases.

Contents

List of Figures

1 Introduction

In the past decade, Peer-to-Peer(P2P) network has become one of the most popular applications over the Internet. We have witnessed a number of successful real-world P2P systems providing content delivery, file synchronization or event Video-on-Demand(VoD) services[1]. Such systems encourage peers to contribute their uploading bandwidth and provide the related incentive algorithms. Therefore, their service capacities are dynamically adjusted by the totally number of peers in the system. This generally incurs better scalability as well as robustness, but the reliability and hence service quality can hardly be guaranteed. There have also been efforts toward synergizing dedicated servers or even cloud-based components with peer-to-peer. Unfortunately, having an all-round scalable, reliable, responsive, and cost-effective solution remains an illusive goal. To make the matter worse, today's P2P systems such as Bittorrent(BT) is widely adopted to deliver large contents over the Internet. The downloading of such contents normally requires longer downloading time. This further increases the risk of content/service unavailable.

From the users' perspective, this is particularly severe especially when they are sharing important data files with a deadline. It is worth noting that the QoS problem in P2P is not limited to its unstable service performance. Due to the dynamic peer arrival and departure, the users cannot even predict their QoS unless they join the swarm and download the entire content. From the service provider's perspective, such a problem greatly affects the commercialization of P2P systems. Using Bittottent as an example, it is known that BitTorrent, Inc. failed to provide BT as a paid content

delivery service, and the company itself was also being sold for 140 million in cash to Justin Sun and his blockchain media startup Tron according to[5]. This is mainly because the users are not willing to buy a service that could suddenly go unavailable for no reason.

It is easy to see that the unpredictable QoS greatly affected the development of P2P systems. To address such a problem, we take initial steps to understand and predict the swarm popularity of Bittorrent systems by exploring its related metainfo. In this thesis, we carefully collect 1250 metainfo files from the torrent sharing sites and use the Enhanced Ctorrent[11] to actively prob their related content feature, peer list as well as their downloading speed. To avoid possible copyright issues, our modified BT client will not upload any contents to the BT swarm. Based on this measurement, we study the file type, the piece length, swarm popularity, and the creation time in the BT system. We also analyze the relationship between each property and the swarm popularity. After that, we apply a neural-network-based approach to predict swarm popularity. Our evaluation shows that the proposed method can provide good accuracy and the loss function converge after iterations.

The remainder of this thesis proceeds as follows. Section II presents an overview of the related works. In Section III, we discuss the details of our measurement configuration. The neural-network-based prediction is presented and evaluated in Section IV and Section V concludes the thesis and discusses potential future directions.

2 Related Works

2.1 Client-Server System

The client-server model is commonly used for a long time. The relationship is usually described as the request and response. There are lots of existing protocols that help to make the connection stable and resilient. Such as the famous TCP/IP protocol. Clients by sending a request on a different port which is chosen by OS and server makes response according to the clients. There are lots of issues that server need to consider such as security issues like preventing the server from DDOS attack or set up firewalls, configure multiple servers in order to balance the requests, the backup plan if the server is down, increase hardware such as network bandwidth, CPU, storage, etc. More and more users start to use applications, the company has to consider the scalability of the system. Most of the big company such as Facebook, Instagram, Uber are using the client-server system, which is easily controlled by the company.

2.2 Peer to Peer System

In the previous section, we introduced the client-server system, the drawback is the central server needs to consider the scalability of the entire system. Comparing with the peer to peer network, there are differences between those two networks. Peer to Peer network is popular and the most important feature is the size of the system

is unlimited, the system can run regardless of the size of the peers. More peers inside the network, the more stability the system has. At the same time, if there is a node malfunctioned, the whole system can still run perfectly.[3] Napster is a Peer to Peer Network file-sharing system like the ancestor of the Peer to Peer Network, there is a central system called directory server inside Napster system, so the central system can know every user connects to itself, whenever the user under the system query for downloading a new music file, the directory server searches the index and return the index back to the user and the connection will be established from the query machine to the other hosting file machines under the directory server.

Gnutella is the larger P2P network, contrast with the Napster, Gnutella doesn't have any centralized server, the client initially finds node inside the system then through that node, the client can connect to other active nodes. After collecting all the nodes information, the client sends a search request to all the nodes, if any of the nodes have that result, those nodes will communicate through UDP and send the searching result back. Compare with Napster, Gnutella is harder to shut down, because there is no central server, if one node or couple nodes inside the network down, the system still can work. However, if the Directory server in Napster down, all the nodes can't find the file index and clients are not able to communicate between themselves.

The BitTorrent protocol creates a swarm of peers and hosts for content distribution. The file is owned by multiple hosts with different segments. After the downloader downloads the whole piece, it can become an uploader, so the BitTorrent protocol works very well under low bandwidth condition. More of the peers inside the swarm means the new peer can access the file from multiple sources which gives better download speed. The peer distributing a data file treats the file as a number of identically sized pieces, usually with byte sizes of a power of 2, the size of each

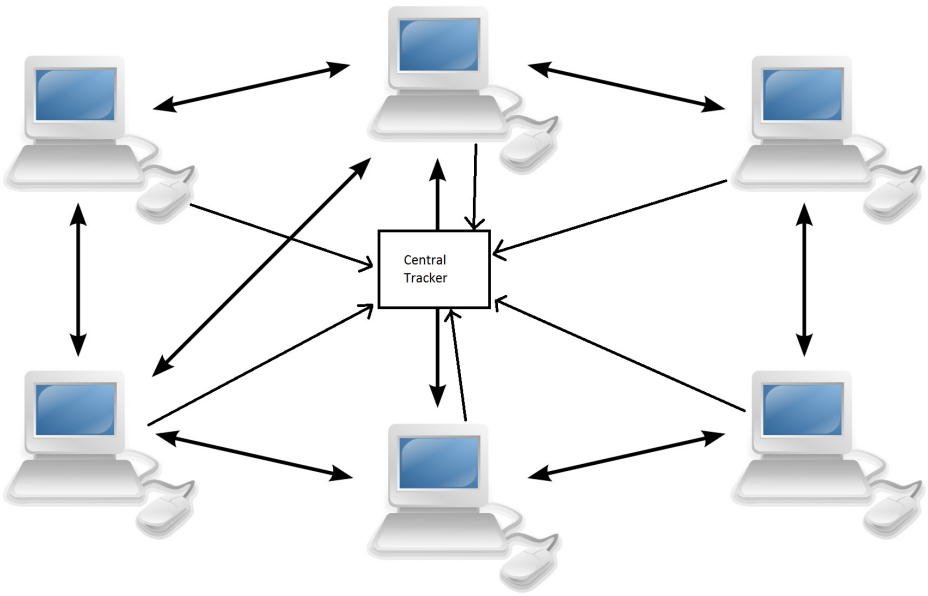

Figure 2.1: Peer To Peer Network with Central Tracker.

piece is fixed in the metadata of the torrent file. The torrent file has a hash for each piece, using the famous hash function, and records the hash value in the torrent file in order to protect the file is not been modified. Pieces with sizes greater than 512MB will reduce the size of a torrent file for a very large payload but is claimed to reduce the efficiency of the protocol.[2] When another peer later receives a particular piece, the hash of the piece is compared to the recorded hash to test that the piece is error-free.[4] The exact information contained in the torrent file depends on the version of the BitTorrent protocol. By convention, the name of a torrent file has the suffix .torrent. Torrent files have an "announce" section, which specifies the URL of the tracker, and an "info" section, containing names for the files, their lengths, the piece length used, the peers' information and seeders information.

BitTorrent algorithm encourages peers to upload the content inside the swarm, at first sort the peers according to their uploading bandwidth (it could be the physical uploading bandwidth or the uploading bandwidth that has been set manually by the

user) such that the peer has the highest uploading bandwidth. The peer selection chooses the peer has the bigger uploader speed. [7]. There are lots of algorithm such as rarest first, random select and optimistic unchoking that keep the healthiness of the entire swarm.

According to the introduction of Peer to Peer network, the peer to peer network helps the central part of the system don't need to consider the scalability of the system. However, The problem of peer to peer network is the performance of the entire system can not be guaranteed, the number of peers inside the swarm is really hard to predict because there is no agreement that all the seeders and peers have to stay inside the swarm to make the contribution. Compare to the client-server system, the server opens up a stable connection with the client, the download speed depends on both server and client-side Internet bandwidth. However, inside the peer to peer network, the new client just added inside the swarm does not have a constant download speed and specific finishing time since the seeders and peers are dynamic. The hypothesis would be if there is a method that can help to predict the popularity of the swarm. So the seeders and peers prefer to stay inside or prefer not to stay, which can potentially indicate the popularity of the swarm. The whole system is dynamic, however, the torrent file properties are stable. As a result, we studied those unchanged properties and make predictions with them and see the results of it.

2.2.1 Other Major Classification Algorithm

There are lots of classification algorithms already implemented by different corporations and research groups. Sklearn provides good resource and built-in functions which gives API for calling. We used Sklearn Logistic regression[10], Decision Tree and Gaussian Naive Bayes to comparing with a neural network prediction results.

Logistic Regression is commonly used for binary classification problem. Because the output is either true or false, so we think it is a good fit in our research. In Sklearn documentation, it is used to predict IRIS dataset. Decision Tree applies to multiple attributes for the data set and build the tree for making the prediction[**DecisionTree1**]. In Sklearn documentation, they used a decision tree on the same IRIS data set for predicting the type of flower. The decision tree chooses on the lowest entropy in order to access higher information gain to better classify the labels and make the prediction. Here is a decision tree example looks like[8].

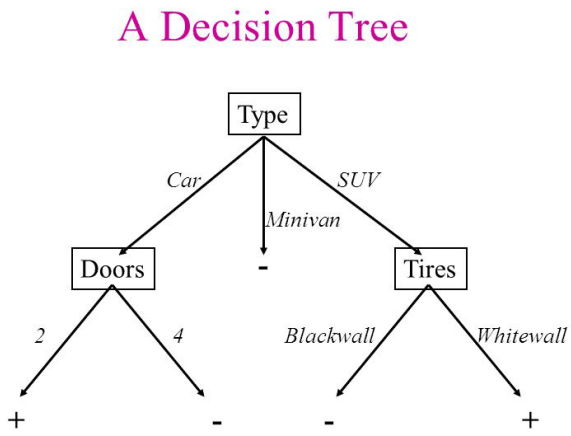

Figure 2.2: Decision Tree

Gaussian Naive Bayes uses Bayes' Theorem that assumes all the features are independent so there are no relationships between two features that depend on each other, the documentation says the continuous values should follow Gaussian distribution[9]. All those classifications belong to supervised learning and included all the results of the classification algorithms.

2.2.2 Neural Network Introduction

A neuron is a cell, which can perform the special function that communicates with another neuron through connections like nerves[6]. Computer scientists borrowed the idea from biology, they learned how the animal brains think and how those neurons work together and give the host idea or some signals to make the host makes some decisions, take image recognition as an example, the human saw the cat image and the eyes trigger the first layer of neurons and the next layer of neurons get output from the previous layer of neurons and trigger the next layer of neurons, etc 2.3. At the final, the human can get the idea of what is the image actually is. And in the real world, the biological brain works much more complex than the description above. However, computer scientists use the idea and implemented the artificial neurons, which are just the numbers, and the weights as known as numbers as well, there are different layers inside neural networks, which is also known as deep learning neural networks.

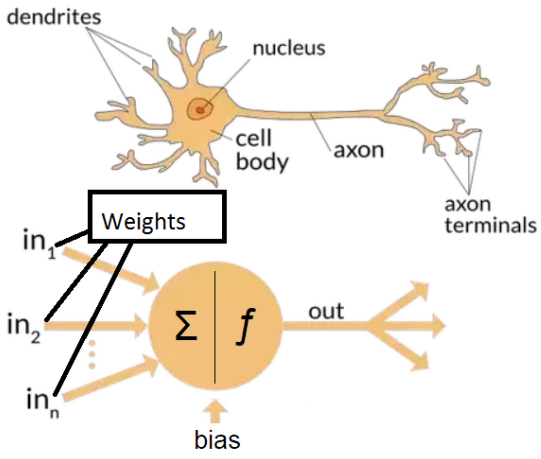

Figure 2.3: Neural Network With Biology Neuron

There are lots of applications are used with neural networks as known as speech

recognition, image recognition, and some classification problems. Basically, there are three different kinds of neural networks used in this research, because the prediction results depend on the parameters such as learning rate, iterations, and mathematics functions. So the best practice for the neural network is to try to use different parameters for the model and see the performance.

2.2.3 Forward Propagation

The neural network can be basically represented by three layers, the input layer, hidden layer, and output layer. The input layer is also called A^0 as for activation function for layer 0. The middle is hidden layers and an output layer. For each training entry, data is feed in to the A^0 and calculated with function $z = w * a + b$, for the matrix expression is $Z = W \cdot A + b$, W is weight matrix and b is bias matrix. As figure 2.3 as example, $Z_1 = W_1 \cdot A_0 + b$and $A_1 = Activation(Z_1)$. There are four different kinds of activation function are commonly be used. The sigmoid function will be mentioned below. Second is the Tanh function, which has ranged from -1 to 1, lots of scientists said the Tanh function is better than Sigmoid as far as classification problems. The rest are ReLU and Leaky ReLU. The most significant reason for using those functions is because those functions are easy to derivative and less mathematical complexity. We are going to introduce the Relu function because it is used after this research. The function is $f(x) = max(0, x)$ and the plot of the function.

The first one is logistic regression, which takes the input and with randomly generated weights and initial as zero bias and calculates the output feed the output into an activation function. The logistic function is used here because the function

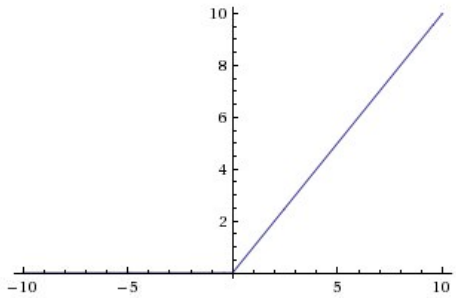

Figure 2.4: Relu Function

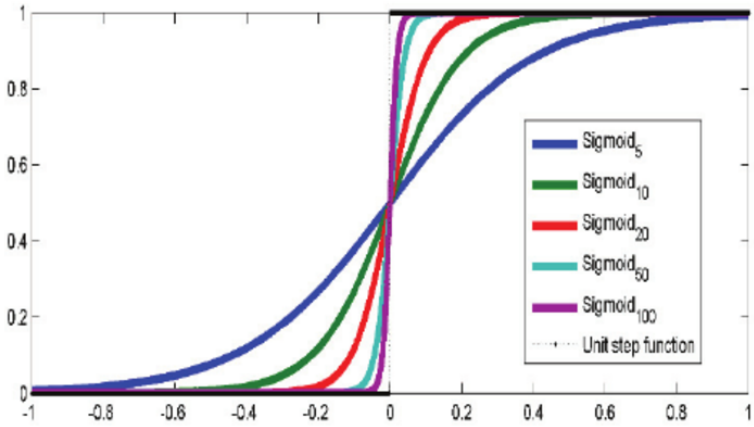

Figure 2.5: Sigmoid Functions Graph

shape is from 0 to 1, the formula is

$$sigmoid(x) = \frac{1}{1 + e^{-}(w^T x + b)} \tag{2.1}$$

With the help of sigmoid function, the prediction stays in between 0 and 1, unlikely linear regression from minus infinity to positive infinity, the activation function takes the result from and maps to a 0 to 1.

$$z = w^T x + b \tag{2.2}$$

This is logistic regression forward propagation. In the neural network, each layer

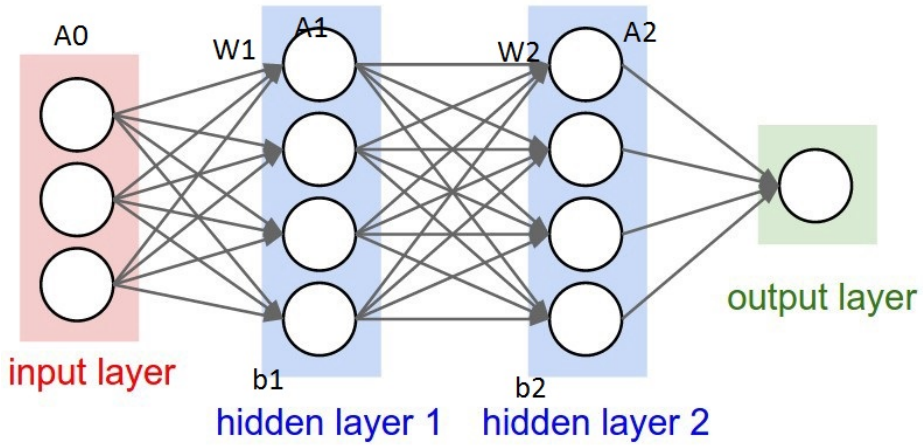

Figure 2.6: Foward Propagation Illustration Graph

takes the input weights and the output from the previous layer then calculating the result and feed through the activation function in order to pass to the next layer. As a result, the forward chain rule is:

$$Z^{[1]} = W^{[1]}X^0 + b^{[1]}$$

$$\begin{aligned}
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^1 + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
Z^{[L]} &= W^{[L]}A^{L-1} + b^{[L]} \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = Y(predict)
\end{aligned} \tag{2.3}$$

2.2.4 Backward Propagation

The backward propagation is learning from the training examples and updating parameters, in order to calculate and update the w, b, need to know dw, db, so the loss function is used here.

$$Loss(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

The a stands for the predicted output and y is training real value. After the loss function is calculated, the $\frac{\partial L}{\partial a}$ is the derivative of activation function. $da = \frac{dL(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$, according to the chain rule and the result is:

$$\frac{\partial L}{\partial a} = -\frac{y}{a} - (-)\frac{1 - y}{1 - a} \tag{2.4}$$

the reason get ∂a is important because the ∂a can compute the derivative of the sigmoid activation function and the sigmoid function is given $a = \frac{1}{1+e^{-}(z)}$, and derivative it.

$$\frac{\partial a}{\partial z} = -(1 + e^{-z})^{-2} * (e^{-z} * (-1)) \rightarrow \tag{2.5}$$

$$\frac{e^{-z}}{(1+e^{-z})^2} \rightarrow$$

$$\frac{1}{1+e^{-z}} * \frac{1+e^{-z}-1}{1+e^{-z}} \rightarrow$$

$$1 - \frac{1}{1+e^{-z}} \rightarrow 1 - sigmoid(z)$$

$$\frac{\partial a}{\partial z} \rightarrow sigmoid(z) * (1 - sigmoid(z))$$

Since $sigmoid(z)$ equals to a, the derivative of sigmoid activation function is $a*(1-a)$. Two formula above have already calculated. $\frac{\partial L}{\partial a}$ and $\frac{\partial a}{\partial z}$. In order to calculate $\frac{\partial L}{\partial z}$, simply multiply the previos two equations up which is $\frac{dL(a,y)}{da} = (-\frac{y}{a} + \frac{1-y}{1-a}) * a(1-a)$. Here we only showing the simplified version of the equation:

$$-y(1-a) + (a)(1-y) \tag{2.6}$$

$$-y + ya + a - ay \rightarrow a - y$$

So far, we can easily get the $\frac{\partial L}{\partial w}$, because follow the chain rule again,

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial w}$$

and the linear function $z = wa + b$ and the $\frac{\partial z}{\partial w} = a$, so the final result of updating weights according to loss function is $\frac{\partial L}{\partial w} = x(a - y)$ and, as far as to the entire training set. Apply the individual to the matrix, the matrix is a combination of all

the individual weights and bias. $\frac{\partial L}{\partial b} = a - y$

$$dz = \frac{dL}{da} * \frac{da}{dz} = a - y \longrightarrow dw = x * dz \longrightarrow dW = \frac{1}{m} X(A - Y)^T$$

$$db = \frac{1}{m} \sum (A - Y)$$

So far, the one iteration of neural network is completed, the forward propagation uses the input and final output, compare with real training set result, calculated the difference, which is cost function, during the back propagation, the w can be updated by $w = w - \eta * \partial w$, the learning rate η is usually a way tell the program how big step is, in order to update the weights. So does bias. The formular to update bias is $b = b - \eta * \partial b$.

We finished all single entry of back propagation. However, in order to increase the efficiency of the neural network calculation time, we have to use vectorization and Python broadcasting, The gradient descent of one hidden layer of neural network listed below, the program implements gradient descent in the later section.

$$\partial Z^{[2]} = A^{[2]} - Y \tag{2.7}$$

$$\begin{aligned}
\partial W^{[2]} &= \frac{1}{m} \partial Z^{[2]} A^{[1]^T} \\
\partial b^{[2]} &= \frac{1}{m} \sum \partial Z^{[2]} \\
\partial Z^{[1]} &= W^{[2]} \partial Z^{[2]} * g^{[1]'}(Z^{[1]}) \\
\partial W^{[1]} &= \frac{1}{m} \partial Z^{[1]} X^T \\
\partial b^{[1]} &= \frac{1}{m} \sum Z^{[1]}
\end{aligned}$$

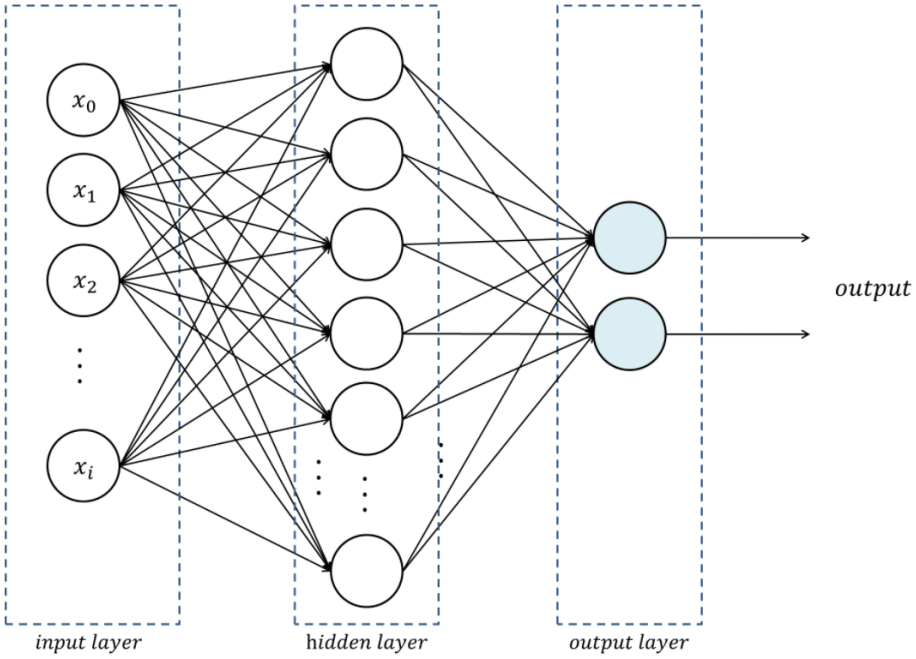

Figure 2.7: One Hidden Layer Neural Network

2.2.5 One hidden Layer Neural Network

The one hidden layer neural network look like figure 2.5. The training X is stacked horizontally. Each value inside each x is multiplied by the weights on the neuron then plus bias, after linear process and activation function, the output from previous layer is new input for next layer.

2.2.6 Deep Neural Network

Compare with the single-layer neural network, the deep neural network has multiple layers of neurons between the input layer and the output layer. However, it has the same components of simple neural networks. First, initialize all the parameters, randomly generate weights variables and set all bias variables to zero. Linear forward function and activation functions. Then calculate the loss function. Through back propagation updates the parameters variables. After all the training sets, the system

is able to predict the final output of the test sets. The same formula is used her instead given a equation:$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$. We use Relu and Sigmoid functions as an activation function for the neural network. $A^{[l]} = activation(Z^{[l]})$. For the backtracking, need to remember all the variables that already calculated. So those calculated variables should be stored inside cache for later back propagation use. The cost function is basically the same. We used the cross-entropy cost function. The activation function backward, the formula to calculate is $\partial Z^{[l]} = \partial A^{[l]} * g^{'}(Z^{[l]})$. As for Relu function, the derivative of function is equals to 0 when X is less than zero and $f^{'}(x) = 1$ when $x > 0$. Back propagation of deep neural network formulas are:

$$\partial A^{[l-1]} = \frac{\partial L}{\partial A^{[l-1]}} = W^{[l]T}\partial Z^{[l]} \tag{2.8}$$

$$\partial W^{[l]} = \frac{\partial L}{\partial W^{[l]}} = \frac{1}{m}\partial Z^{[l]}A^{[l-1]T} \tag{2.9}$$

$$\partial b^{[l]} = \frac{\partial L}{\partial b^{[l]}} = \frac{1}{m}\sum \partial Z^{[l]} \tag{2.10}$$

3 Measurement and Configuration

3.1 Collecting Torrent Files

We used python Scrapy project to get the dataset which is powerful tool go over the data on web pages and extracts useful data. First we installed the Scrapy on Ubuntu and specify the website that spider goes.

```
1 import scrapy
2
3
4 class PirateBayeSpider(scrapy.Spider):
5     name = "pirateBay_spider"
6     start_urls = ['https://www.pirate bay.net/recent']
```

According to the HTML code on the pirate bay website, we extract the information. Because the HTML code is huge, the one listed down is just an example, scrapy can extract the information from tag. Here is file type as an example.

```
1     <center>
2       <a href="/browse/200" title="More from this category">Video</a><br
            >
3       <a href="/browse/205" title="More from this category">TV shows</a>
4     </center>
```

We scraped the magnet links from the website, then the file stores the recent added torrent file's magnet links. Here is the main code from the Scrapy project. The code under is the main spider code and stores the results to CSV file after the first page is scanned, it goes to next page by adding the URL with page number and keep scanning. Except for those torrent files on the website and different pages, we

added some very popular ISO torrent. Totally we got 1250 torrent files.

```
import scrapy


class Piratebayscraper(scrapy.Spider):
    name = 'pirateBay_spider'
    allowed_domains = ['https://www.pirate-bay.net/recent']
    start_urls = ['https://www.pirate-bay.net/recent']

    custom_settings={ 'FEED_URI': "thepirate-bay_%(page)s.csv),
                      'FEED_FORMAT': 'csv'}

    def parse(self, response):
        CATEGORY_SELECTOR=response.css('<center>.//dl[dt/text() = "title
            "]/dd/a/text()').extract()
        NAME_SELECTOR=response.css('.value::title').extract()
        DATE_SELECTOR=response.xpath("//em[@class='detDesc']/text()").
            extract()

        row_data=zip(category, name, date)

        for item in row_data:
            scraped_info = {
                'page':response.url,
                'category' : item[0],
                'name' : item[1],
                'date' : item[2],
            }

        NEXT_PAEG = '<a href="/recent">:attr(href)</a>'
        next_page = response.css(NEXT_PAGE).extract_first()
        if next_page:
            yield scrapy.Request(
                response.urljoin(next_page),
                callback=self.parse
            )
}
```

3.2 CTorrent-based Measurement

Enhanced C is a torrent client was written by C, which is a BitTorrent client under Linux environment, basiclly it is a open source software brings clients into BitTorrent protocol. The whole code is really long and here we only show the parts that we

modified in order to get the useful information for our research. First we turned off the upload speed because it is illegal to upload the content without copyright and school ITSS would not allow the research. Here is the code to set the upload.

```
1     case 'f':         // force seed mode, skip sha1 check when startup.
2       arg_flg_force_seed_mode = 1;
3       break;
4
5     case 'D':         // download bandwidth limit
6       cfg_max_bandwidth_down = (int)(strtod(optarg, NULL) * 1024);
7       break;
8
9     case 'U':         // upload bandwidth limit
10      //cfg_max_bandwidth_up = (int)(strtod(optarg, NULL) * 1024);
11      cfg_max_bandwidth_up = 0;
12      break;
13
14    case 'P':         // peer ID prefix
15      l = strlen(optarg);
16      if (l > MAX_PF_LEN) {
17        CONSOLE.Warning(1, "-P arg must be %d or less characters",
              MAX_PF_LEN);
18        return -1;
19      }
20      if (l == 1 && *optarg == '-') *arg_user_agent = (char) 0;
21      else strcpy(arg_user_agent,optarg);
22      break;
```

Next step we obtained all the features associate with this torrent file and generate the output with download speed, trackers Ips, peers inside the swarm, etc. The program opens the file called "Stats.txt" and write the information into it. Enhanced ctorrent program starts with ctorrent.cpp and the main function has functions to analysis the input parameters from user, creates file to store the object file and connect with tracker. Then call Downloader object starts to download file. This is the part that ctorrent starts to connect to peers and start to download.

```
1     if( Tracker.Initial() < 0 ){
2       CONSOLE.Warning(1, "error, tracker setup failed.");
3       exit(1);
4     }
5
6     sig_setup();  // setup signal handling
```

```
7       CONSOLE.Interact(
8         "Press 'h' or '?' for help (display/control client options)." );
9       Downloader();
10      if( cfg_cache_size ) BTCONTENT.FlushCache();
11    }
12    if( !arg_flg_exam_only ) BTCONTENT.SaveBitfield();
13    WORLD.CloseAll();
```

Peerlist.cpp contains peers adding and deleting method for new peer coming and leaving from the swarm. So we found the function and add the code in order to get the IP information from the clients.

```
1       peer->SetConnect();
2       peer->SetAddress(addr);
3       peer->stream.SetSocket(sk);
4       peer->SetStatus( (-2 == r) ? P_CONNECTING : P_HANDSHAKE );
5       //write peer IP address to file
6       FILE * pfile;
7       pfile = fopen("Stats.txt","a");
8       fprintf(pfile, "Peer IP: %s\n", inet_ntoa(addr.sin_addr));
9       fclose(pfile);
10
11      if(arg_verbose) CONSOLE.Debug("Connecting to %s:%hu (peer %p)",
12          inet_ntoa(addr.sin_addr), ntohs(addr.sin_port), peer);
13
14    }else{
15      if( setfd_nonblock(sk) < 0) goto err;
16
17      peer = new btPeer;
```

BtContent.cpp contains information about the torrent file such as Meta data, created date, piece length, we insert the code inside BtContent in order to capture those data.

```
1     FILE * pfile;
2     pfile = fopen("Stats.txt","a");
3
4     CONSOLE.Print("META INFO");
5     CONSOLE.Print("Announce: %s", m_announce);
6     if( m_announcelist[0] ){
7       CONSOLE.Print("Alternates:");
8       for( int n=0; n < 9 && m_announcelist[n]; n++ )
9         CONSOLE.Print(" %d. %s", n+1, m_announcelist[n]);
10    }
11    if( m_create_date ){
```

```
12      char s[42];
13  #ifdef HAVE_CTIME_R_3
14      ctime_r(&m_create_date, s, sizeof(s));
15  #else
16      ctime_r(&m_create_date, s);
17  #endif
18      if( s[strlen(s)-1] == '\n' ) s[strlen(s)-1] = '\0';
19      CONSOLE.Print("Created On: %s", s);
20
21      //write Date into a file
22      //fwrite(s, 1, strlen(s),pfile);
23      fprintf(pfile, "Created on: %s\n", s);
24    }
25    CONSOLE.Print("Piece length: %lu", (unsigned long)m_piece_length);
26    //write Piece length to file
27    fprintf(pfile, "Piece length: %lu\n", (unsigned long)m_piece_length);;
28    fclose(pfile);
```

In Console.cpp, we modified the code to get download rate and peers count. There are some torrent files doesn't work, when we run batch execution, we specified the 20s as total waiting time for a torrent. We defined the execution variable to get download speed at after the number of execution. Download speed and peers count can call the function inside Tracker and BTcontent.

```
1   //wirte the peers count and download speed to the file;
2     execution++;
3     if(execution == 5){
4     FILE * pfile;
5     pfile = fopen("Stats.txt","a");
6     fprintf(pfile, "Download rate: %d K/s\n", (int)(Self.RateDL() >> 10));
7     fprintf(pfile, "Peers count: %d\n\n", (int)(Tracker.GetPeersCount()));
8     fclose(pfile);
9     execution++;
10  }
11    if((int)(Self.RateDL() >> 10) > 0 && wroten == false){
12        FILE * pfile;
13        pfile = fopen("Stats.txt","a");
14        fprintf(pfile, "Download rate: %d K/s\n", (int)(Self.RateDL() >>
              10));
15        fprintf(pfile, "Download start up after: %d s\n", (int)(now-
              BTCONTENT.GetStartTime()));
16        fprintf(pfile, "Peers count: %d\n\n", (int)(Tracker.GetPeersCount
              ()));
17        fclose(pfile);
18        wroten = true;
19    }
```

```
20   FILE * pfile;
21   pfile = fopen("Stats.txt","a");
22   fprintf((pfile), "Download time used: %d s\n", (int)(now-BTCONTENT.
         GetStartTime()));
23   fclose(pfile);
24   if((int)(now-BTCONTENT.GetStartTime()) == 19){
25       FILE * pfile;
26       pfile = fopen("Stats.txt","a");
27       fprintf((pfile), "The torrent still not start download after 19s,
             torrent doesn't work!\n");
28       fclose(pfile);
29   }
```

3.3 Measurement Results

Here is portion of the result of the code after running. We listed 3 torrents for example.

```
1  Created on: Sun Apr 19 10:01:39 2015
2  Piece length: 2097152
3  Total: 1128 MB
4  Created on: Thu Jul 26 11:54:31 2018
5  Piece length: 524288
6  Total: 812 MB
7  Peer IP: 81.171.22.66
8  Peer IP: 46.188.49.98
9  Peer IP: 66.189.77.71
10 Peer IP: 213.188.245.139
11 Peer IP: 91.149.211.112
12 Peer IP: 77.161.214.210
13 Peer IP: 195.154.163.119
14 Peer IP: 91.240.66.215
15 Peer IP: 186.50.40.199
16 Peer IP: 23.237.0.42
17 Peer IP: 31.208.161.46
18 Peer IP: 212.51.10.28
19 Peer IP: 208.188.185.12
20 Peer IP: 87.122.252.151
21 Peer IP: 74.105.93.225
22 Peer IP: 93.200.7.60
23 Peer IP: 185.157.160.63
24 Peer IP: 185.21.216.152
25 Peer IP: 173.239.230.11
26 Peer IP: 213.93.16.185
27 Peer IP: 122.116.104.21
```

```
Peer IP: 68.190.112.125
Peer IP: 134.249.190.45
Peer IP: 80.109.139.34
Peer IP: 202.153.213.68
Peer IP: 88.133.15.111
Peer IP: 90.186.241.28
Peer IP: 192.99.10.67
Peer IP: 177.54.158.219
Peer IP: 73.95.105.107
Peer IP: 109.228.53.87
Peer IP: 77.188.21.147
Peer IP: 5.100.41.227
Peer IP: 83.84.168.75
Peer IP: 176.9.37.132
Peer IP: 188.192.2.15
Peer IP: 150.246.31.54
Peer IP: 64.42.179.43
Peer IP: 188.234.82.69
Peer IP: 93.104.212.253
Peer IP: 192.131.44.91
Peer IP: 89.212.17.215
Peer IP: 45.37.181.76
Peer IP: 166.70.84.193
Peer IP: 51.15.177.148
Peer IP: 185.21.216.146
Peer IP: 54.36.63.109
Peer IP: 213.152.161.15
Peer IP: 145.236.27.13
Peer IP: 71.65.204.221
Download rate: 16270 K/s
Download start up after: 2 s
Peers count: 1572

Created on: Tue Jul 12 13:04:41 2016
Piece length: 1048576
Total: 1021 MB
Peer IP: 37.231.164.182
Peer IP: 92.96.235.181
Peer IP: 149.147.14.123
Peer IP: 176.202.185.187
Peer IP: 185.45.195.194
Peer IP: 131.212.41.237
Download rate: 427 K/s
Download start up after: 2 s
Peers count: 6

Created on: Fri Jul  8 03:10:18 2016
Piece length: 1048576
Total: 4064 MB
Peer IP: 131.212.41.237
The torrent still not start download after 20s, torrent doesn't work!
```

```
79 Peers count: 1
```

After we got those results, we converted the txt file into CSV file in order to make data manipulation. Input file as Stat1.txtt and output as Stats1.csv. The result of CSV file (partial).

```
1  with open("Stats.txt", 'r') as inFile, open("Stats1.csv", 'w') as
       outFile:
2      outFile.write("Created Date,Piece Length,File Length,Initial Download
           Rate,Start Download After,Peers Count\n")
3      for line in inFile:
4          if 'Created on: ' in line:
5              line = line.rstrip("\n\r")
6              line = line.replace('Created on: ', '')
7              outFile.write(line+",")
8              continue
9          if 'Piece length: ' in line:
10             line = line.rstrip("\n\r")
11             line = line.replace('Piece length: ', '')
12             outFile.write(line+",")
13             continue
14         if 'Total: ' in line:
15             line = line.rstrip("\n\r")
16             line = line.replace('Total: ', '')
17             line = line.replace(' MB', '')
18             outFile.write(line+",")
19             continue
20         if 'Download start up after: ' in line:
21             line = line.rstrip("\n\r")
22             line = line.replace('Download start up after: ', '')
23             line = line.replace(' s', '')
24             outFile.write(line+",")
25             continue
26         if 'Download rate: ' in line:
27             line = line.rstrip("\n\r")
28             line = line.replace('Download rate: ', '')
29             line = line.replace(' K/s', '')
30             outFile.write(line+",")
31         if 'The torrent still not start download' in line:
32             outFile.write("0,40,")
33         if 'Peers count: ' in line:
34             line = line.replace('Peers count: ', '')
35             outFile.write(line)
36             continue
37 inFile.close()
38 outFile.close()
```

```
Created Date,Piece Length,File Length,Initial Download Rate,Start
    Download After,Peers Count,
Thu Mar  9 21:41:32 2017,2097152,2721,132,2,6,
Tue Mar 14 08:22:28 2017,1048576,646,170,9,2,
Fri Dec 23 08:43:32 2016,4194304,3007,16,4,2,
Tue Feb 28 10:59:25 2017,16384,0,16,2,2,
Mon Feb 27 04:03:25 2017,262144,397,9,3,4,
Fri Feb 24 23:01:45 2017,524288,288,5,5,3,
Fri Mar 10 07:12:56 2017,1048576,1272,110,3,3,
```

3.4 Shell Script Control Flow

The Linux shell script helps to control the Enhanced C torrent runs for 20 seconds, because for some torrents, there is no peers inside the swamp and the consequence is the rest of the torrent inside the directory can not be executed and analyzed. The shell script is easy, does two things, traverse through the directory and kill the process after 20 seconds.

```
#!/bin/bash
FILES=/home/bo/Downloads/ctorrent-dnh3.3.2/torrents/*.torrent
for f in $FILES
do
    ctorrent -s "$f-result" "$f" &
    sleep 20; kill $!
done

}
```

As the conclusion of the data, we keep the total downloaders under 100 as unpopular torrent and total larger or equal 100 as the popular torrent. We encoded as 0 and 1 as the result. Even though we already got the data, the prediction is still challenging, users always changing inside the swarm, user can join and leave in anytime they want. Also, there is no such math formula can perfect fit the properties to the result. And we discard the uncertain facts, we used a neural network to predict because, under big neural network application such as the facial recognition, the

input values seem random but actually using the neural network, the different input actives different neurons and the results sometimes better than expected.

4 Neural Network Based Framework

4.1 Framework Configuration

4.1.1 Build The Logistic Regression

First part of the research, we built logistic regression. According the Sigmoid function, the code is:

```
1 def sigmoid(z):
2
3     s = 1/(1+np.exp(-z))
4
5     return s
6 }
```

The numpy library gives great implementation of *exp*.

Logistic regression is a neural network with only one neural, and the output is 0 or 1, If the result is 0, which means the torrent is bad torrent according to the attributes, if the output is 1, which means it is popular. During the data prepossessing, we evaluated the peers count if the count is over 100 is popular torrent. If the peers count is less than 100. It is a non popular torrent.

Weights and bias arguments are initialized with "0".

```
1 def initialize_weights_and_bias(dim):
2
3     w = np.zeros((dim, 1))
```

```
    b = 0
    return w, b
```

As the single forward propagation, we can get the cost function and the first predict with initialized weights and bias.

```
def propagate_cal(w, b, X, Y):

    m = X.shape[1]
    cost = (-1/m)*np.sum(Y*np.log(A)+(1-Y)*np.log(1-A))

    dw = 1/m*np.dot(X,(A-Y).T)
    db = 1/m*np.sum((A-Y))
    cost = np.squeeze(cost)
    grads = {"dw": dw,
             "db": db}
    return grads, cost
```

For the number of iterations, each iter can update the weights and bias through dw and db and lost function. After 3000 iteration, the accuracy is really high up to 68%.

4.1.2 Deep Neural Network

Deep neural network has same procedure as the logistic regression above. For the initialize parameters part, we passed the array with $1 * n$ dimension for each layer neurons it has. With weights are initialized by random numbers and bias are zero. Under the help of numpy random. Variable "parameter" is hashmap structure, so the key is the string based from $w1...wn$ and $b1...bn$.

```
def initialize_parameters_deep(layer_dims):
    np.random.seed(3)
    parameters = {}
    L = len(layer_dims)

    for l in range(1, L):

```

```
        parameters['W' + str(l)] = np.random.randn(layer_dims[l],
            layer_dims[l-1])*0.01
        parameters['b' + str(l)] = np.zeros((layer_dims[l],1))


    return parameters
}
```

The linear forward is same as single layer networks. However, in order to use gradient descent in future, we stored the A from previous layer and W, b.

```
def linear_forward(A, W, b):

    Z = np.dot(W,A)+b
    cache = (A, W, b)

    return Z, cache
```

Inside the neuron, the activation function we used was Relu and the final layer is Sgimoid functiom. The Linear activation forward is below, with L different layers:

```
def linear_activation_forward(A_prev, W, b, activation):
    if activation == "sigmoid":

        Z, linear_cache = linear_forward(A_prev, W, b)
        A, activation_cache = sigmoid(Z)

    elif activation == "relu":

        Z, linear_cache = linear_forward(A_prev, W, b)
        A, activation_cache = relu(Z)


    cache = (linear_cache, activation_cache)

    return A, cache


def L_model_forward(X, parameters):

    caches = []
    A = X  #The input layer
    L = len(parameters) # number layers in the neural network

    for l in range(1, L):
        A_prev = A

```

```
27          A, cache = linear_activation_forward(A_prev, parameters['W'+str(
                l)], parameters['b'+str(l)], activation="relu")
28          caches.append(cache)
29
30
31      AL, cache = linear_activation_forward(A, parameters['W'+str(L)],
            parameters['b'+str(L)], activation="sigmoid")
32      caches.append(cache)
33
34
35      return AL, caches
```

Now the new cache includes the parameters from linear foward W, b, A, but the expression here A is the result after activation function.

According to the cost function, the cost function is implemented below:

```
1  def compute_cost(AL, Y):
2
3      m = Y.shape[1]
4
5      cost = -1/m*np.sum(np.multiply(Y, np.log(AL))+np.multiply((1-Y), np.
           log(1-AL)))
6
7
8      return cost
```

After getting loss function, the back propagation follow the formula introduced in the background. Linear backward can get $dW, dbanddA$ as long as known dZ.

```
1  def linear_backward(dZ, cache):
2
3      A_prev, W, b = cache
4      m = A_prev.shape[1]
5
6
7      dW = 1/m*np.dot(dZ, A_prev.T)
8      db = np.sum(dZ, axis = 1, keepdims=True)/m
9      dA_prev = np.dot(cache[1].T,dZ)
10
11     return dA_prev, dW, db
```

Put the activation derivative and linear together can iterative get previous layer.

```
def L_model_backward(AL, Y, caches):

    grads = {}
    L = len(caches) # the number of layers
    m = AL.shape[1]
    Y = Y.reshape(AL.shape)

    dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))

    current_cache = caches[-1]
    grads["dA" + str(L-1)], grads["dW" + str(L)], grads["db" + str(L)] =
        linear_backward(sigmoid_backward(dAL, current_cache[1]),
       current_cache[0])

    for l in reversed(range(L-1)):
        current_cache". Outputs: "grads["dA" + str(l)] , grads["dW" +
            str(l + 1)] , grads["db" + str(l + 1)]

        current_cache = caches[l]
        dA_prev_temp, dW_temp, db_temp = linear_backward(relu_backward(
            grads["dA" + str(l + 1)], current_cache[1]),current_cache
            [0])
        grads["dA" + str(l)] = dA_prev_temp
        grads["dW" + str(l + 1)] = dW_temp
        grads["db" + str(l + 1)] = db_temp

    return grads
```

Then update the parameters for deep neural network.

```
def update_parameters(parameters, grads, learning_rate):

    L = len(parameters)

    for l in range(L):
        parameters["W" + str(l+1)] = parameters["W" + str(l + 1)] -
            learning_rate * grads["dW" + str(l + 1)]
        parameters["b" + str(l+1)] = parameters["b" + str(l + 1)] -
            learning_rate * grads["db" + str(l + 1)]
```

At the last, we put all the functions together and make the prediction.

```
def L_layer_model(X, Y, layers_dims, learning_rate = 0.0075,
    num_iterations = 3000, print_cost=False):
np.random.seed(1)
    costs = []
    parameters = initialize_parameters_deep(layers_dims)

```

```
    for i in range(0, num_iterations):

        [LINEAR -> RELU]*(L-1) -> LINEAR -> SIGMOID.

        AL, caches = L_model_forward(X, parameters)

        cost = compute_cost(AL, Y)

        grads = L_model_backward(AL, Y, caches)

        parameters = update_parameters(parameters, grads, learning_rate)

        if print_cost and i % 100 == 0:

        if print_cost and i % 100 == 0:
            costs.append(cost)

    plt.plot(np.squeeze(costs))
    plt.ylabel('cost')
    plt.xlabel('iterations (per hundreds)')
    plt.title("Learning rate =" + str(learning_rate))
    plt.show()

    return parameters
```

4.2 Eveluation

We analyzed from one of the famous torrent file IP results and used Python Geoip2 library to change IP addresses to geolocation and marked on the map. Here is the result around the world. From the geological map shows that Europe countries and North South American are highly using Torrent protocol. We tested during the daytime on CT, probably that is the potential problem.

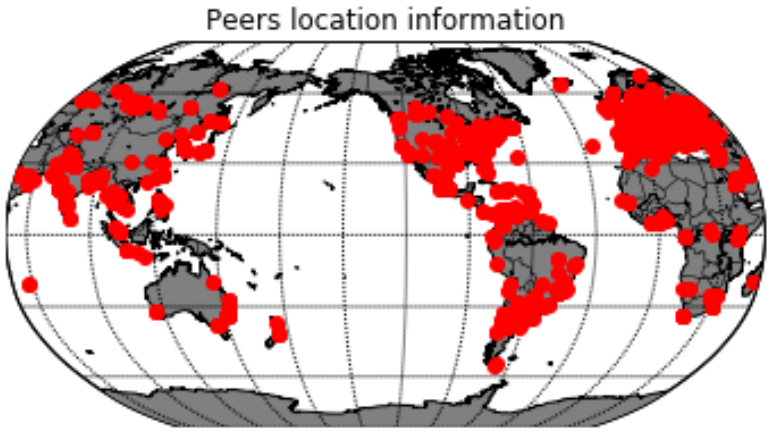

Figure 4.1: IP Geolocation

Graph 4.2 shows relationship between file type and their popularity, system big image files are most popular over the samples. Graph 4.3 shows all of samples, the

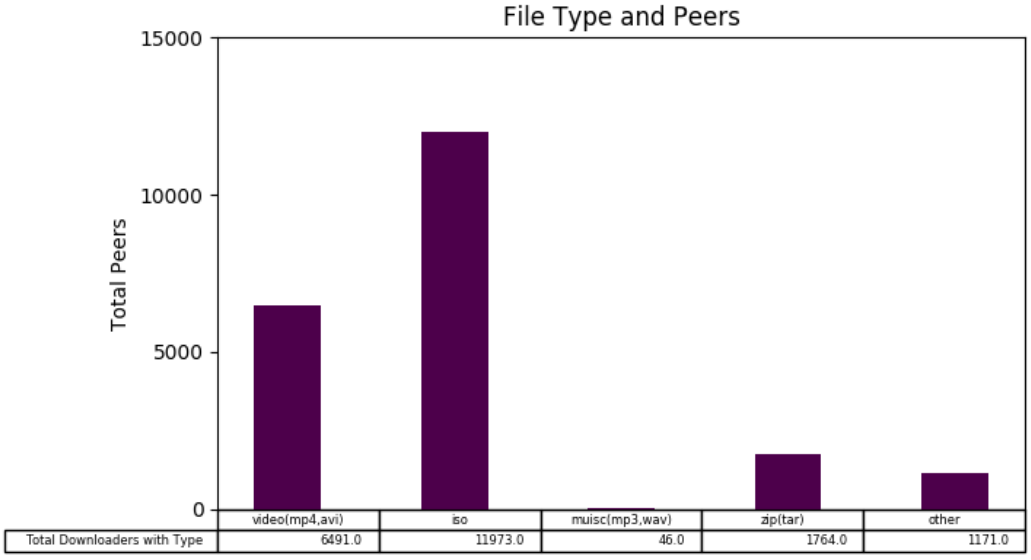

Figure 4.2: File Type and Popularity.

length 262144B which is 0.25MB are the most popular size, 1MB is the second. After

convention, the rest of peice length are 2MB and 4MB.There are other piece length however, too many to list on the graph so we put all the others into the "other" category. Graph 4.4 gives pie chart for better illustration.

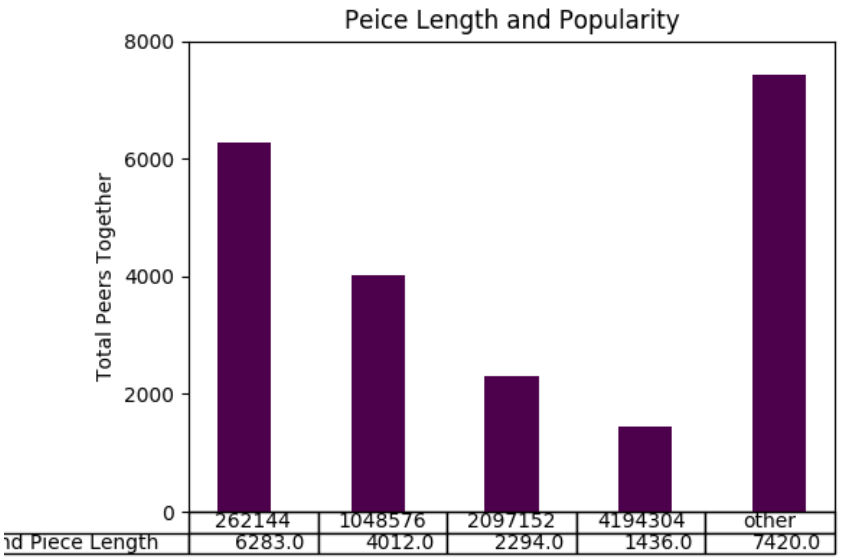

Figure 4.3: Piece Length and Popularity.

From the pie chart graph we can see 0.25MB piece length and other size of piece length takes total around 60% of the total downloaders. So there can be roughly concluded that most popular piece length is 0.25MB. However, this creates lots of uncertainty because the other category takes a big part as well.

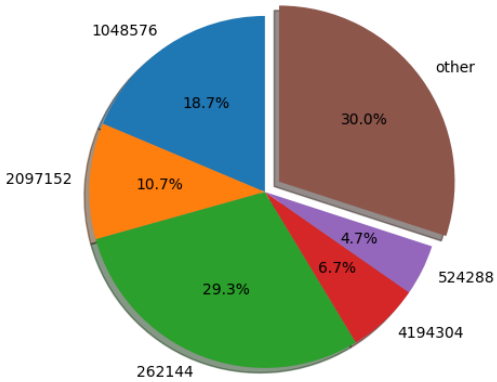

Figure 4.4: Piece Length and Popularity Pie Chart.

Grapgh 4.5 shows almost all of the samples are during the year of 2019, and the older torrent the less polularity. From graph 4.6 shows 2019 takes almost all the peers compare to the other 3 years. We didn't include the year before 2015 because most of the torrent files doesn't work. And it will create too much noisy data for the sample data. For year of 2019 we used the torrent created the first half of the year. The year and popularity relationship kind of like exponential increase.

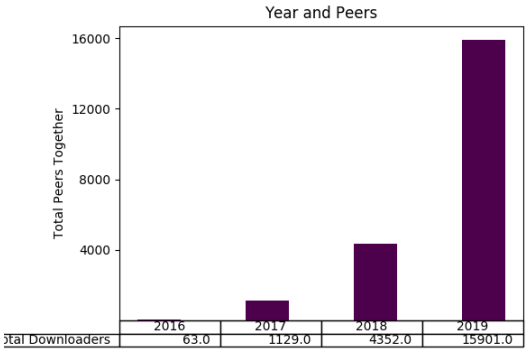

Figure 4.5: Years and Popularity.

As the conclusion of pie chart, we can easily see the torrent popularity is highly related to the time and the seeders usually doesn't seed old torrent and the total downloaders are very small. Most of old torrent files doesn't work.

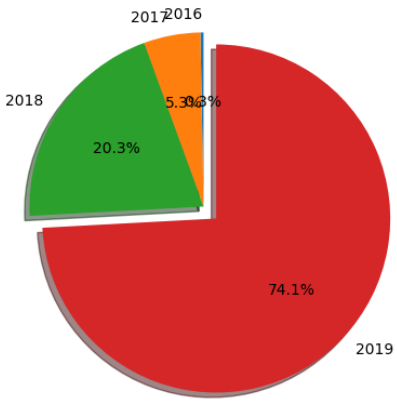

Figure 4.6: Years and Popularity Pie Chart.

From graph 4.7, we can see the file size around 0 GB up to 10GB are really popular. There are some downloaders for the torrent file bigger than 10GB but most of users download files under 100GB.

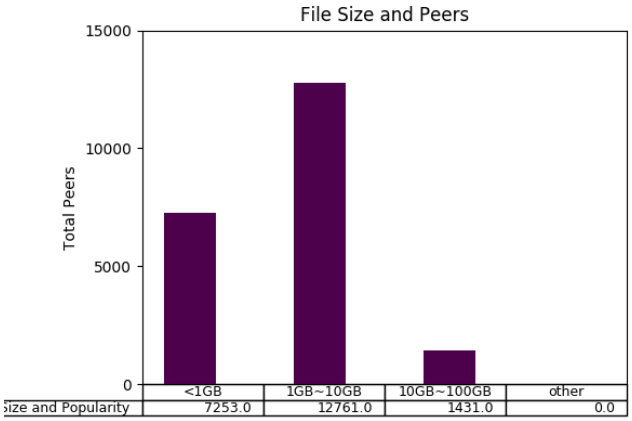

Figure 4.7: File Size and Popularity

We success got 1250 their properties and the popularity relationships, we analyzed the features from the data. First we used all the classification algorithms from Sklearn, which includes Logestic regression, decision tree and Gaussian Naive Bayes, we split all the samples into $4:1$ as taining and testing data and we ran the library 5 times to get the results, from graph 4.8 we can tell except the neural network, other algorithm doesn't quite fit the training set.

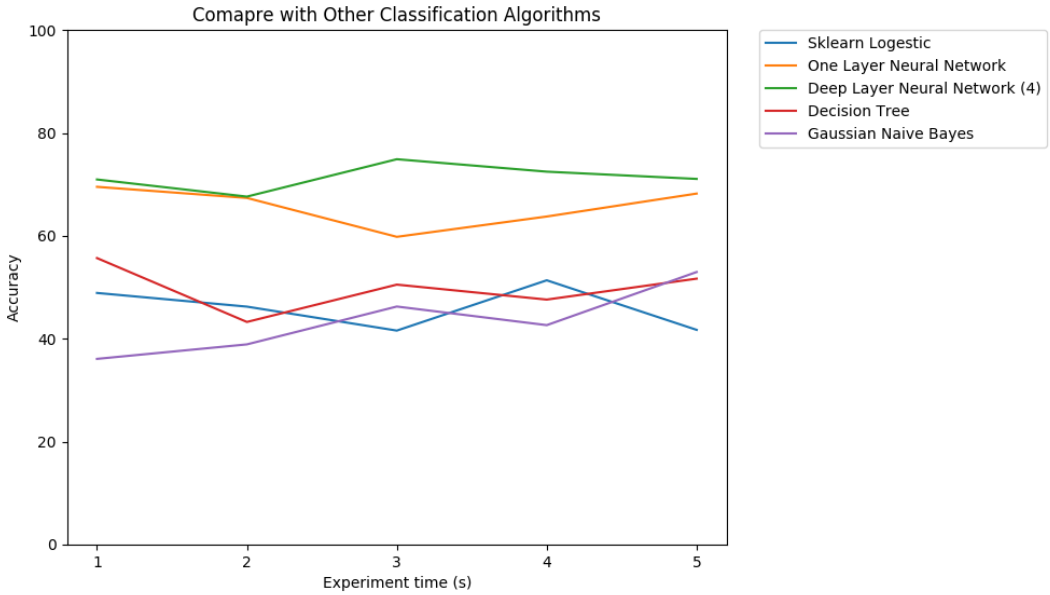

Figure 4.8: Experiment time and Accuracy

Graph 4.9 shows the average and standard deviation of the result, which shows deep neural network gives better result less standard deviation, one layer neural network is the second place, decision tree predicts better than the other two. However the average value of decision tree is around 50% so it is just around randomly guess.

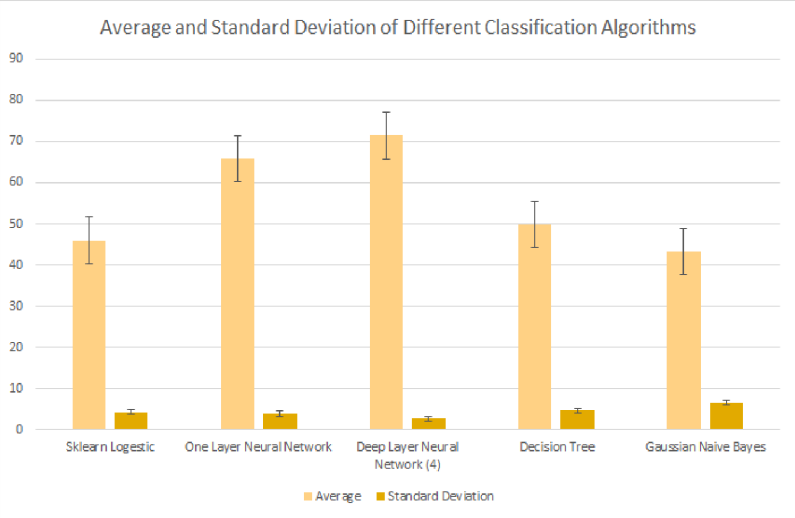

Figure 4.9: Average and STD in different Model

For the one layer neural network with learning rate is 0.005, the cost function is showed below 4.10 which has minimal cost compare with other learning rate value. The best accuracy for one hidden layer is 72.64%, the reason why we used only one layer because for the image prediction, the larger layer neural network can give better result, so we using one layer compare to four layers and see the difference. And it works better here too.

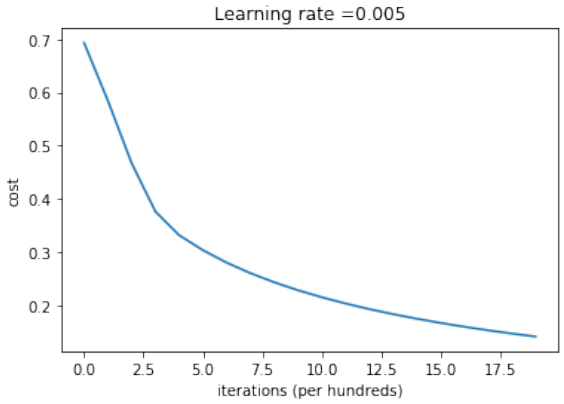

Figure 4.10: One hidden neural network Cost after Iterations.

For the different learning rate, the cost function value after hundreds iterations showed in 4.11. When the learning rate is too small, the loss function doesn't converge too well. The change of parameters are too small to make the final function converge. The graph shows when learning rate is equals to 0.01, the loss function converge better than the other two values. When learning rate equals 0.0001, the loss function after 1400 hundred times of iteration still keeps above 0.6.

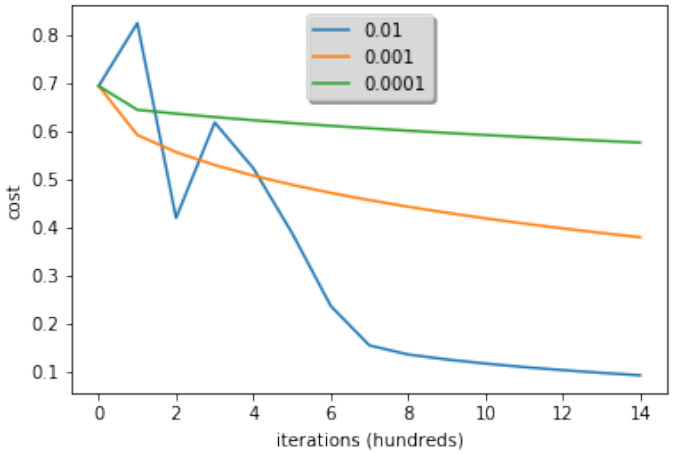

Figure 4.11: Different Learning Rate and Accuracy.

Deep neural network with 4 layer model. And we used 0.005 and 0.0075 as different

learning rate. The best testing accuracy is 74.91%. As the graph 4.12 and 4.13 shows that both of them converge good.We used 0.0075 as the final experiment value.

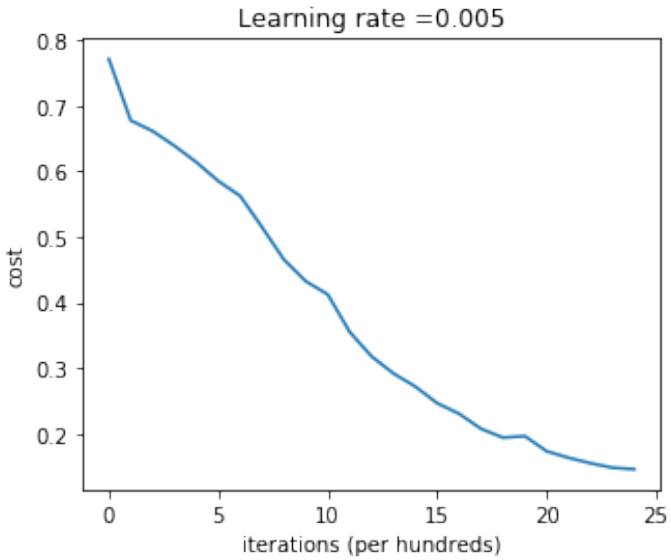

Figure 4.12: Deep neural network learning rate = 0.005.

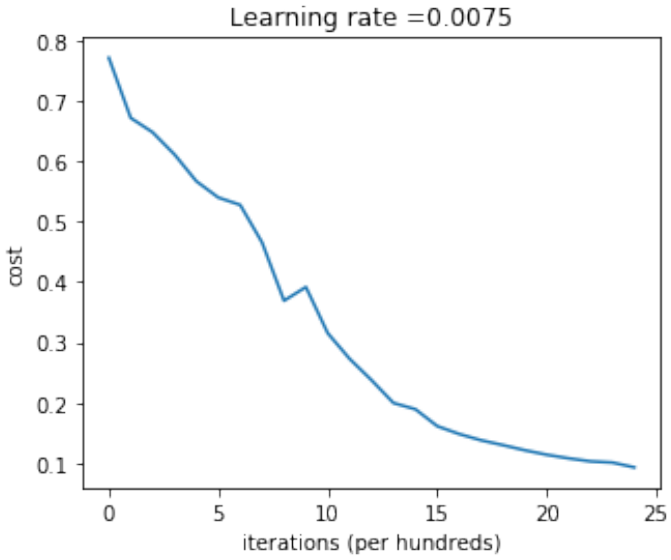

Figure 4.13: Deep Neural Network learning rate = 0.0075.

As the conclusion, we got the result with deep neural network worked out as best.

The result wasn't too exciting because the prediction wasn't good. Only successfully predicted 75% of samples. Maybe there was some problems with data prepossessing and the sample selections. Also after turning off the uploading speed, the popularity we measured was not same as without turning it off, which is a grea impact of the experiments.

5 Conclusions and Future Work

Throughout this research, we built the neural network takes on the torrent files' properties predict the torrent files popularity. As a result, neural networks can make better predictions compare with other classification algorithms. However, the result is kind of predictable but still hard to perfectly figure out the popularity. Because the torrent's popularity is affected by lots of other facts too.

This research can help people study about the torrent and gives network service providers some information about how to configure their network in order to control or understand the torrent, also clients prefer to see the popularity before actually download the torrent file and join inside the swarm.

For the future work, we can add more features into the neural network such as the tracker information, other seeders' information for better prediction accuracy. Since we turned off the uploading speed of the Ctorrent client for legal issues, future we can configure the VPN and use VPN to study relationships between properties and popularity would be helpful. Also Kruskal algorithm can connect all the nodes by building a minimum spanning tree. Under the helping of Kruskal's algorithm, we can connect all the peers inside the swarm and calculate the total distance of the whole tree. Maybe the smaller distance can give faster download speed. At the last, maybe add those features inside the prediction, will definitely impact our existing results and would be interesting to see when it works out.

References

[1] D. Barkai. *Peer-to-peer computing : technologies for sharing and collaborating on the net. Hillsboro*. 2001 (cit. on p. 1).

[2] “”BitTorrent Specification””. In: Wiki.theory.org., 2012 (cit. on p. 5).

[3] S. N. Choon Hoong Ding and R. Buyya. “Peer-to-Peer Networks for Content Sharing”. In: The University of Melbourne, Australia: cloudbus. URL: http://www.cloudbus.org/papers/P2PbasedContentSharing.pdf (cit. on p. 4).

[4] B. Cohen. “The BitTorrent Protocol Specification”. In: 2008 (cit. on p. 5).

[5] K. Lam. *PWinning bidder of Warren Buffettfffdfffdfffds private lunch revealed as Tron CEO Justin Sun*. 7 June 2019. URL: https://www.foxbusiness.com/business-leaders/winning-bidder-warren-buffett-private-lunch-revealed-justin-sun,https://techcrunch.com/2018/06/18/bittorrent-tron/ (cit. on p. 2).

[6] R. PA. *Neuronal excitability: voltage-dependent currents and synaptic transmission*. April 1992 (cit. on p. 8).

[7] D. Qiu and R. Srikant. “Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks”. In: University of Illinois at Urbana-Champaign: ACM, 2004, pp. 367–378 (cit. on p. 6).

[8] J. R. Quinlan. *Simplifying decision trees". International Journal of Man-Machine Studies*. 2013 (cit. on p. 7).

[9] J. Rennie. *Tackling the poor assumptions of Naive Bayes classifiers*. 2003 (cit. on p. 7).

[10] D. Walker SH; Duncan. *Estimation of the probability of an event as a function of several independent variables*. 1967 (cit. on p. 6).

[11] J. Wang Liang; Kangasharju. *Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT*. 15 May 2014. URL: https://www.cs.helsinki.fi/u/lxwang/publications/P2P2013_13.pdf (cit. on p. 2).